ACCESO A DATOS

UT2 - MANEJO DE CONECTORES

PRÁCTICA TURSO

AUTORES: Vincenzo Rojas Carrera y Diego de Haro Ruiz

CICLO FORMATIVO DE GRADO SUPERIOR DE DESARROLLO DE APLICACIONES MULTIPLATAFORMA

1. Introducción

Desarrolla una aplicación de gestión de información (el tema puede elegirse libremente, tras aprobación del profesor, para evitar temas repetidos), que permite almacenar, consultar, modificar y eliminar datos en una base de datos SQLite en cloud.

Nuestro proyecto tiene como objetivo desarrollar una aplicación de gestión de información para un **juego de rol**, permitiendo almacenar, consultar, modificar y eliminar datos relacionados con ubicaciones, personajes, enemigos y objetos..

2. Objetivos del proyecto

- Implementar **CRUD completo** para todas las entidades del juego: ubicaciones, personajes, enemigos y objetos.
- Realizar **consultas complejas**, incluyendo agregaciones, uniones y ordenaciones.
- Sincronizar los datos con **Turso** para almacenamiento remoto en cloud.
- Comparar las ventajas e inconvenientes de usar SQLite local frente a Turso remoto.
- Documentar todas las decisiones de diseño y explicar la estructura del código y de la base de datos.

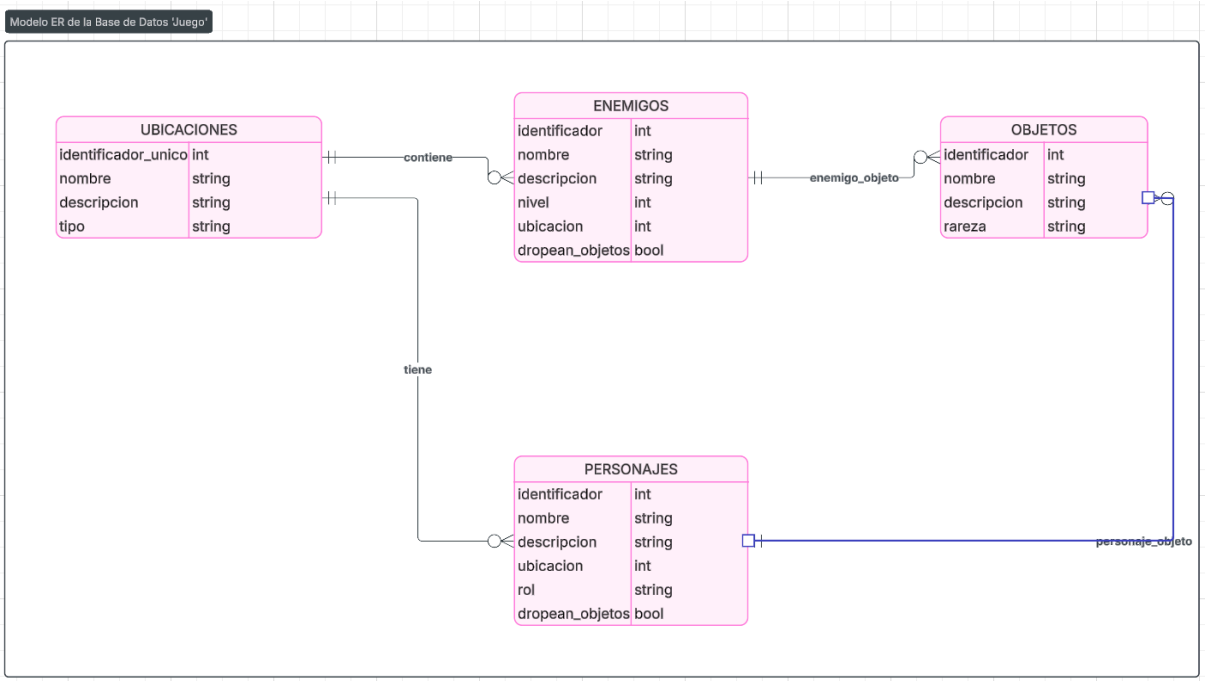
3. Diseño de la aplicación

Modelo de datos

El modelo se ha diseñado para permitir consultas avanzadas y reflejar las relaciones reales entre entidades:

- **Ubicaciones:** identificador único, nombre, descripción y tipo (ciudad, mazmorra, mundo).
- **Personajes:** identificador, nombre, descripción, ubicación, rol y si dropean objetos.
- **Enemigos:** identificador, nombre, descripción, nivel, ubicación y si dropean objetos.
- **Objetos:** identificador, nombre, descripción, rareza y relación opcional con personajes o enemigos que los dropean.
- **Relaciones muchos-a-muchos:** *personaje_objeto* y *enemigo_objeto* para reflejar que múltiples personajes o enemigos pueden llevar o dropear varios objetos.

Esquema de la base de datos



4. Implementación

Estructura de carpetas y archivos

- Carpeta principal: *practicaUT2-AAD-git/*
    - *.gitignore*
    - *README*
    - *bd_conn.py*
    - *consultas_complejas_1.py*
    - *consultas_complejas_2.py*
    - *crud_enemigos.py*
    - *crud_objetos.py*
    - *crud_personajes.py*
    - *crud_ubicaciones.py*
    - *insercion_in*icial.py
    - menu.py

- Subcarpetas:
    - *docs/: memoria.*

Conexión y sincronización con Turso

- Se utilizaron **variables de entorno** (*TURSO_URL*, *TURSO_TOKEN*) gestionadas con *envyte*.
- Conexión mediante *libsql.connect()* y sincronización con `conn.sync()`.
- Se implementó una función genérica *ejecutar_query(query)* para ejecutar cualquier consulta, con manejo de errores y logs.

Funciones CRUD y consultas

- Todas las entidades tienen funciones CRUD separadas (*crud_ubicaciones.py*, *crud_personajes.py*, *crud_enemigos.py* y *crud_objetos.py*).
- Consultas complejas y reportes se encuentran en *consultas_complejas_1.py* y *consultas_complejas_1.py*.
- El menú de consola (*menu.py*) permite navegar y ejecutar comandos CRUD.

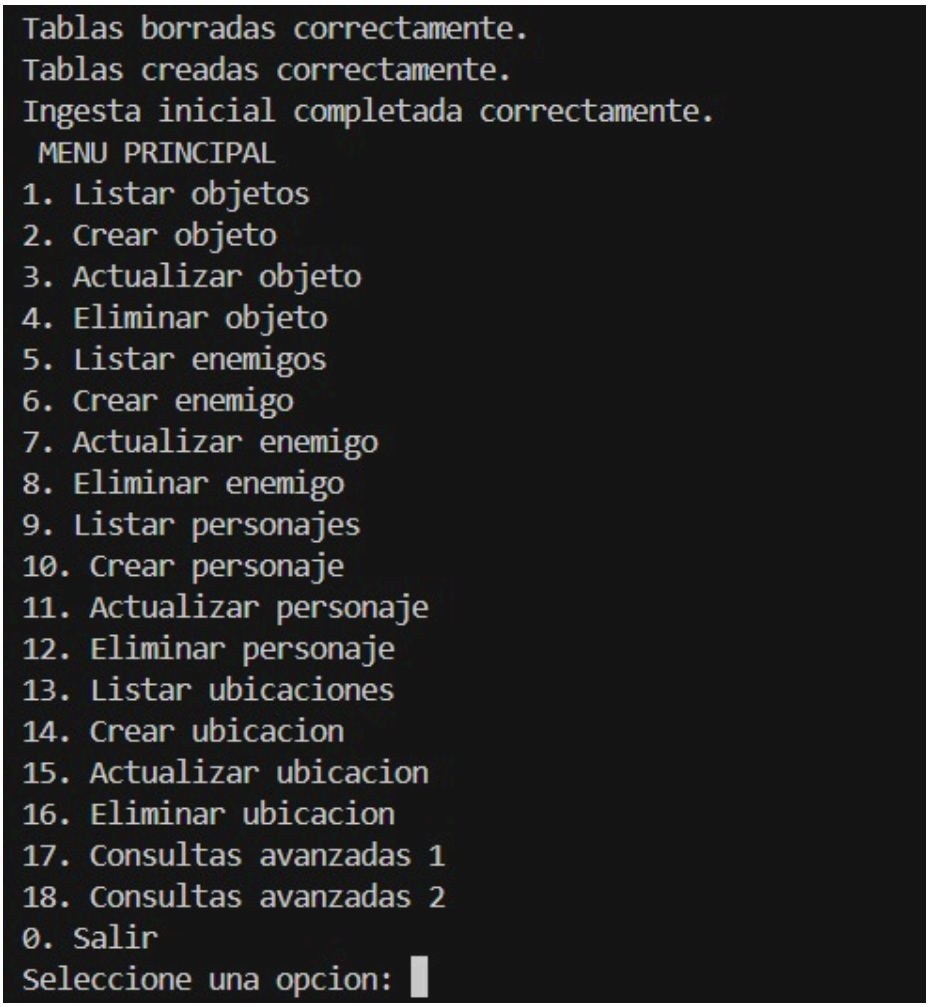
5. Ingesta inicial de datos

- Primero se borran las tablas si existen, luego se crean nuevamente.
- Se insertan datos de prueba para **ubicaciones, personajes, enemigos y objetos**.
- Esta secuencia garantiza que la base de datos en Turso esté sincronizada con la versión local de prueba.

6. Tutorial de uso de la aplicación

Para iniciar la aplicación, abre una consola o terminal en la carpeta del proyecto y ejecuta el siguiente comando: *python menu.py*

Al ejecutarse, se mostrará el siguiente menú principal:



Selecciona una opción introduciendo su número correspondiente y presiona **Enter**.

Las opciones del 1 al 16 permiten gestionar los distintos elementos (objetos, enemigos, personajes y ubicaciones).

Las opciones **17** y **18** ejecutan consultas avanzadas para generar reportes y estadísticas.

La opción **0** cierra la aplicación.

Gestión de Objetos

Al ejecutar la aplicación y seleccionar una de las opciones del menú principal relacionadas con **Objetos** (opciones 1 a 4), el usuario puede realizar distintas operaciones sobre los registros almacenados en la base de datos. Las funciones disponibles son las siguientes:

1. **Listar objetos:** muestra en pantalla todos los objetos registrados, indicando su identificador, nombre, tipo y demás atributos relevantes.

```
Seleccione una opcion: 1
Lista de objetos
(1, 'Espada de Madera', 'Espada basica', 'comun', 2, None, 1)
(2, 'Amuleto Raro', 'Amuleto con poderes', 'raro', None, 3, 2)
(3, 'Pocion de Vida', 'Recupera salud', 'comun', 1, 1, None)
```

2. **Crear objeto:** permite añadir un nuevo registro solicitando los datos necesarios (nombre, descripción, tipo, valor, etc.).

```
Seleccione una opcion: 2
Nombre: Espada_Romboide
Descripcion: Espada con forma de rombo
Rareza: comun
```

```
Lista de objetos
(1, 'Espada de Madera', 'Espada basica', 'comun', 2, None, 1)
(2, 'Amuleto Raro', 'Amuleto con poderes', 'raro', None, 3, 2)
(3, 'Pocion de Vida', 'Recupera salud', 'comun', 1, 1, None)
(4, 'Espada_Romboide', 'Espada con forma de rombo', 'comun', None, None, None)
```

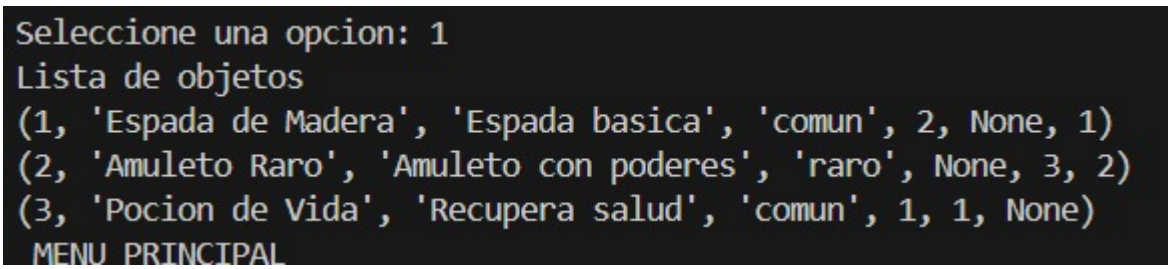
3. **Actualizar objeto:** posibilita modificar la información de un objeto existente. El usuario selecciona el objeto por su identificador y actualiza los campos deseados.

```
Seleccione una opcion: 3
ID del objeto: 4
Nuevo nombre (enter para omitir): Espada_Circular
Nueva descripcion (enter para omitir): Espada con forma circular
Nueva rareza (enter para omitir): raro
```

```
Seleccione una opcion: 1
Lista de objetos
(1, 'Espada de Madera', 'Espada basica', 'comun', 2, None, 1)
(2, 'Amuleto Raro', 'Amuleto con poderes', 'raro', None, 3, 2)
(3, 'Pocion de Vida', 'Recupera salud', 'comun', 1, 1, None)
(4, 'Espada_Circular', 'Espada con forma circular', 'raro', None, None, None)
```

4. **Eliminar objeto:** elimina un objeto de la base de datos tras una confirmación, evitando eliminaciones accidentales.

```
Seleccione una opcion: 4
ID del objeto: 4
```



Cada una de estas operaciones proporciona mensajes informativos sobre el resultado de la acción (éxito o error), garantizando una interacción clara y segura con los datos.

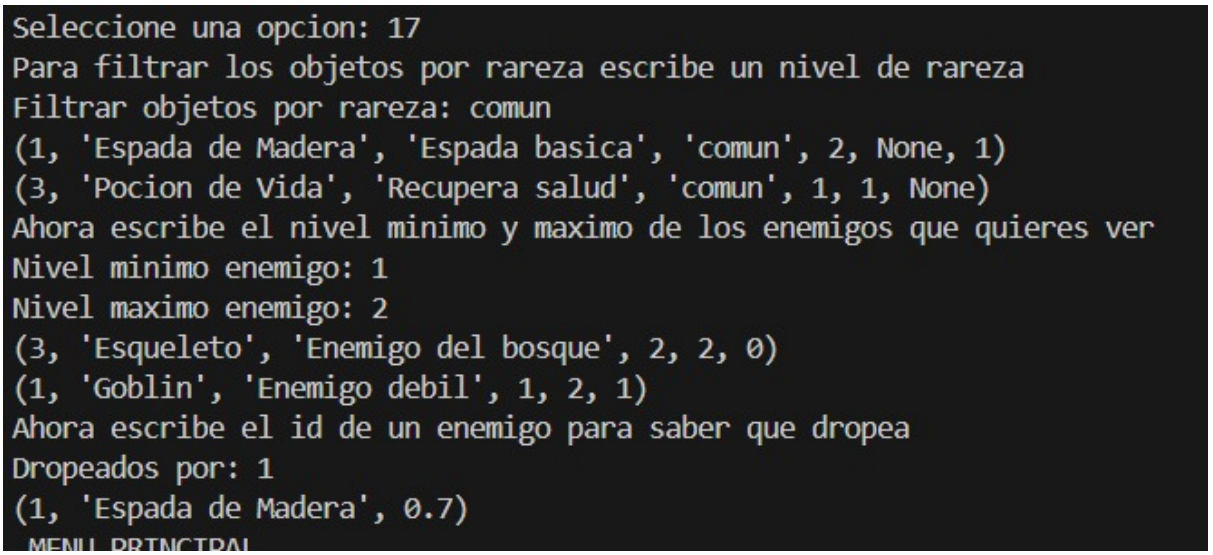
Gestión de las demás tablas

El funcionamiento descrito para la tabla **Objetos** se aplica del mismo modo a las tablas **Enemigos**, **Personajes** y **Ubicaciones** (opciones 5 a 16 del menú principal).

En cada caso, el usuario dispone de opciones para **listar, crear, actualizar y eliminar** registros, siguiendo una estructura y comportamiento uniformes que facilitan la gestión coherente de todos los elementos del sistema.

Consultas avanzadas – Opción 17

La opción 17 del menú principal permite ejecutar **consultas complejas** sobre la base de datos para generar reportes y obtener información más detallada de los elementos gestionados por la aplicación.



Las consultas disponibles son:

1. **Objetos por rareza:**
   Permite listar todos los objetos de un determinado nivel de rareza, ordenados alfabéticamente por nombre. El usuario proporciona el valor de rareza deseado y la aplicación muestra los resultados correspondientes.

2. **Enemigos por nivel:**
   Devuelve todos los enemigos cuyo nivel se encuentra dentro de un rango especificado por el usuario. Los resultados se ordenan de mayor a menor nivel,

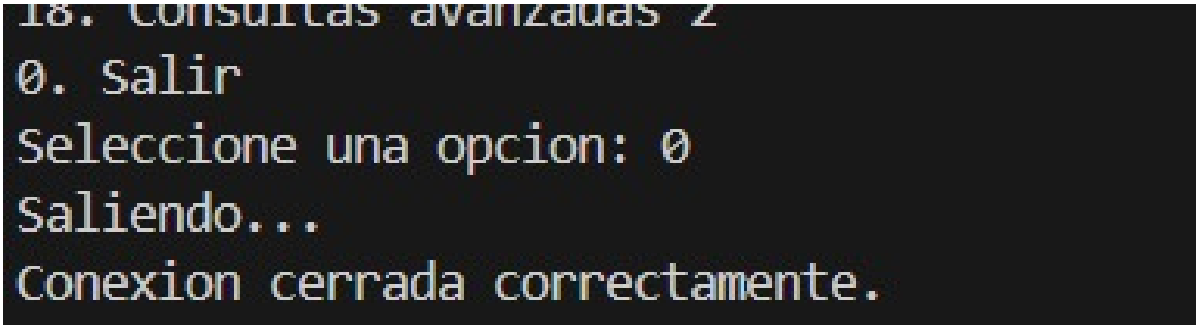
facilitando la visualización de los enemigos más poderosos dentro del rango indicado.

3. **Objetos dropeados por enemigo:**
   Muestra los objetos que puede soltar un enemigo concreto, incluyendo la probabilidad de que cada objeto sea obtenido. El usuario selecciona el enemigo mediante su identificador y recibe la lista de objetos asociados.

Estas consultas proporcionan información resumida y filtrada que permite generar **estadísticas y reportes útiles** para la gestión avanzada de los datos. De manera similar funciona la opción 18.

Salida de la aplicación – Opción 0

La opción 0 del menú principal permite **cerrar la aplicación** de manera controlada y segura.



Al seleccionar esta opción:

- Se finalizan todas las operaciones en curso y se liberan los recursos utilizados por la aplicación, incluyendo conexiones abiertas a la base de datos.

- Se muestra un mensaje de confirmación indicando que la aplicación se ha cerrado correctamente.

- No se permite continuar con ninguna otra operación hasta que la aplicación se vuelva a ejecutar.

Esta opción garantiza que los datos permanezcan **consistentes y protegidos**, evitando pérdidas o inconsistencias que podrían producirse si se interrumpiera la ejecución de forma abrupta. El funcionamiento de cierre es uniforme para todas las sesiones de la aplicación, independientemente de las operaciones previas realizadas por el usuario.

7. Consultas complejas y reportes

- Consultas realizadas en consultas_complejas_1.py:

    - Listado de enemigos por nivel descendente.
    - Objetos agrupados por rareza y dropeo.
    - Estadísticas de cuántos objetos tiene cada personaje.

- Consultas realizadas en consultas_complejas_2.py:

    - Listado de enemigos por nivel descendente.
    - Objetos agrupados por rareza y dropeo.
    - Estadísticas de cuántos objetos tiene cada personaje.
    - Conteo de personajes por ubicación.
    - Listado de personajes que otorgan objetos.
    - Ubicaciones con mayor cantidad de personajes (top 5).

8. Comparativa SQLite local vs Turso

En este proyecto se ha empleado Turso como única base de datos, y se realiza una comparación con SQLite local para analizar sus diferencias y justificar la elección.

SQLite local destaca por su rapidez y simplicidad en entornos donde la aplicación y la base de datos se ejecutan en la misma máquina. Es una opción ideal para desarrollo y pruebas, ya que ofrece acceso directo a los datos y compatibilidad total con las funciones nativas de SQLite. Sin embargo, su uso se limita al equipo donde se almacena el archivo de la base de datos, lo que dificulta el acceso desde otros dispositivos y la colaboración entre varios usuarios. Además, la sincronización de datos entre entornos debe realizarse manualmente.

Turso, en cambio, utiliza el motor libSQL y ofrece almacenamiento en la nube, accesible desde cualquier lugar con conexión a Internet. Esto facilita el trabajo colaborativo y garantiza que los datos estén siempre actualizados y centralizados. Aunque puede presentar una ligera latencia respecto a SQLite local y ciertas restricciones en funciones avanzadas del motor original, su capacidad de sincronización automática y su accesibilidad remota compensan ampliamente esas limitaciones.

9. Bibliografía / Referencias

- Documentación de Turso y libsql: https://turso.tech/docs
- Github: github